# **Looping Techniques in Python**

Python supports various looping techniques by certain inbuilt functions, in various sequential containers. These methods are primarily very useful in competitive programming and also in various projects which require a specific technique with loops maintaining the overall structure of code.  A lot of time and memory space is been saved as there is no need to declare the extra variables which we declare in the traditional approach of loops.

### **Where they are used?**

Different looping techniques are primarily useful in the places where we don’t need to actually manipulate the structure and order of the overall containers, rather only print the elements for a single-use instance, no in-place change occurs in the container. This can also be used in instances to save time.

### **Different looping techniques using Python data structures  are:**

* **Using enumerate():**enumerate() is used to loop through the containers printing the index number along with the value present in that particular index.

|  |
| --- |
| # python code to demonstrate working of enumerate()    forkey, value inenumerate(['The', 'Big', 'Bang', 'Theory']):      print(key, value) |

**Output:**

0 The

1 Big

2 Bang

3 Theory

|  |
| --- |
| # python code to demonstrate working of enumerate()    forkey, value inenumerate(['Geeks', 'for', 'Geeks',                               'is', 'the', 'Best',                               'Coding', 'Platform']):      print(value, end=' ') |

**Output:**

Geeks for Geeks is the Best Coding Platform

* **Using zip():**zip() is used to combine 2 similar containers(list-list or dict-dict) printing the values sequentially. The loop exists only till the smaller container ends. A detailed explanation of zip() and enumerate() can be found [here](https://www.geeksforgeeks.org/using-iterations-in-python-effectively/).

|  |
| --- |
| # python code to demonstrate working of zip()    # initializing list  questions =['name', 'colour', 'shape']  answers =['apple', 'red', 'a circle']    # using zip() to combine two containers  # and print values  forquestion, answer inzip(questions, answers):      print('What is your {0}?  I am {1}.'.format(question, answer)) |

**Output:**

What is your name? I am apple.

What is your color? I am red.

What is your shape? I am a circle.

* **Using iteritem():** iteritems() is used to loop through the dictionary printing the dictionary key-value pair sequentially which is used before Python 3 version.
* **Using items():** items() performs the similar task on dictionary as iteritems() but have certain disadvantages when compared with iteritems().
* It is **very time-consuming**. Calling it on large dictionaries consumes quite a lot of time.
* It takes a **lot of memory**. Sometimes takes double the memory when called on a dictionary.

**Example 1:**

|  |
| --- |
| # python code to demonstrate working of items()    d ={"geeks": "for", "only": "geeks"}    # iteritems() is renamed to items() in python3  # using items to print the dictionary key-value pair  print("The key value pair using items is : ")  fori, j ind.items():      print(i, j) |

**Output:**

The key value pair using iteritems is :

geeks for

only geeks

The key value pair using items is :

geeks for

only geeks

**Example 2:**

|  |
| --- |
| # python code to demonstrate working of items()    king ={'Akbar': 'The Great', 'Chandragupta': 'The Maurya',          'Modi': 'The Changer'}    # using items to print the dictionary key-value pair  forkey, value inking.items():      print(key, value) |

**Output:**

Akbar The Great

Chandragupta The Maurya

Modi The Changer

* **Using sorted():**sorted() is used to print the **container is sorted order**. It **doesn’t sort the container** but just prints the container in sorted order for 1 instance. The use of **set() can be combined to remove duplicate** occurrences.

**Example 1:**

|  |
| --- |
| # python code to demonstrate working of sorted()    # initializing list  lis =[1, 3, 5, 6, 2, 1, 3]    # using sorted() to print the list in sorted order  print("The list in sorted order is : ")  fori insorted(lis):      print(i, end=" ")    print("\r")    # using sorted() and set() to print the list in sorted order  # use of set() removes duplicates.  print("The list in sorted order (without duplicates) is : ")  fori insorted(set(lis)):      print(i, end=" ") |

**Output:**

**The li**st in sorted order is :

1 1 2 3 3 5 6

The list in sorted order (without duplicates) is :

1 2 3 5 6

**Example 2:**

|  |
| --- |
| # python code to demonstrate working of sorted()    # initializing list  basket =['guave', 'orange', 'apple', 'pear',            'guava', 'banana', 'grape']    # using sorted() and set() to print the list  #  in sorted order  forfruit insorted(set(basket)):      print(fruit) |

**Output:**

apple

banana

grape

guava

guave

orange

pear

* **Using reversed():**reversed() is used to print the values ofthe **container in the reversed order**. It does not reflect any changes to the original list

**Example 1:**

|  |
| --- |
| # python code to demonstrate working of reversed()    # initializing list  lis =[1, 3, 5, 6, 2, 1, 3]      # using reversed() to print the list in reversed order  print("The list in reversed order is : ")  fori inreversed(lis):      print(i, end=" ") |

**Output:**

The list in reversed order is :

3 1 2 6 5 3 1

**Example 2:**

|  |
| --- |
| # python code to demonstrate working of reversed()    # using reversed() to print in reverse order  fori inreversed(range(1, 10, 3)):      print(i) |

**Output:**

7

4

1

* These techniques are quick to use and reduce coding effort. for, while loops need the entire structure of the container to be changed.
* These Looping techniques do not require any structural changes to the container. They have keywords that present the exact purpose of usage. Whereas, no pre-predictions or guesses can be made in for, while loop i.e not easily understand the purpose at a glance.
* Looping technique makes the code more concise than using for & while looping.